NPS-CAG-14-009

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

Trusted Computing Exemplar: Quality Assurance Plan

by

Paul C. Clark, Cynthia E. Irvine, and Thuy D. Nguyen

12 December 2014

Approved for public release; distribution is unlimited

Prepared for: United States Navy, OPNAV N2/N6

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Ronald A. Route
President

Douglas A. Hensler
Provost

The report entitled “Trusted Computing Exemplar: Quality Assurance Plan” was prepared for United States Navy, OPNAV N2/N6 and funded in part by United States Navy, OPNAV N2/N6.

Further distribution of all or part of this report is authorized.

This report was prepared by:

Paul C. Clark
Research Associate

Cynthia E. Irvine
Distinguished Professor

Thuy D. Nguyen
Research Associate

Reviewed by:

Cynthia E. Irvine, Chair
Cyber Academic Group

Released by:

Jeffrey D. Paduan
Dean of Research

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE ***(DD-MM-YYYY)***
12-12-2014

2. REPORT TYPE
Technical Report

3. DATES COVERED ***(From-To)***
Nov 2013 to Nov 2014

4. TITLE AND SUBTITLE
Trusted Computing Exemplar: Quality Assurance Plan

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Paul C. Clark, Cynthia E. Irvine, and Thuy D. Nguyen

5d. PROJECT NUMBER
W4C05

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-CAG-14-009

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rhonda Onianwa
OPNAV, N2N6 F13
rhonda.onianwa@navy.mil

LT David Rivera
OPNAV, N2/N6F1
david.j.rivera4@navy.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for pubic release; distribution is unlimited

13. SUPPLEMENTARY NOTES
The view expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense of the U.S. Government.

14. ABSTRACT

This document describes the Life Cycle Management Plan for the development of a high assurance secure product. A high assurance product is one for which its users have a high level of confidence that its security policies will be enforced continuously and correctly. Such products are constructed so that they can be analyzed for these characteristics. Lifecycle activities ensure that the product reflects the intent to ensure that the product is trustworthy and that vigorous efforts have been made to ensure the absence of unspecified functionality, whether accidental or intentional.

In particular, this document expands and unifies the testing requirements that are stated in the *Life Cycle Management Plan*, the *Configuration Management Plan*, and the *Software Development Standards*.

This Quality Assurance (QA) Plan emphasizes requirements, restrictions, standards, responsibilities, etc., for these required tests. Specifically excluded from this plan, however, are the formal and semi-formal work, code correspondence, and covert channel analysis. In addition, there will need to be independent re-testing and penetration testing performed. It is also recognized that quality means more than just source code testing (such as conformance to documentation standards, correct spelling, etc.); those issues are currently covered in other documents.

15. SUBJECT TERMS
Machinery control systems, MCS, life cycle security, high assurance, system security, trustworthy systems

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	UU	23	Cynthia E. Irvine
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** ***(include area code)*** (831) 656 2461

THIS PAGE INTENTIONALLY LEFT BLANK

CYBER ACADEMIC GROUP

NAVAL POSTGRADUATE SCHOOL

NPS-CAG-14-009

Trusted Computing Exemplar: Quality Assurance Plan

Paul C. Clark
Cynthia E. Irvine
Thuy D. Nguyen

December 2014

ACKNOWLEDGEMENT

The authors gratefully acknowledge the following organizations for providing support toward the development of this work: OPNAV N2/N6 F1.

The material presented here builds upon work supported in previous years by the Office of Naval Research.

A portion of the material presented here is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 and CNS-0430598. This document does not necessarily reflect the views of the National Science Foundation.

Table of Contents

Table of Figures

This document describes a preliminary design for quality assurance that was not executed within the research project that generated the document.

1 Introduction

This document has been written in support of a research project to publicly demonstrate and document how a high assurance product can be developed and distributed. A high assurance product is one for which its users have a high level of confidence that its security policies will be enforced continuously and correctly. Such products are constructed so that they can be analyzed for these characteristics. Lifecycle activities ensure that the product reflects the intent to ensure that the product is trustworthy and that vigorous efforts have been made to ensure the absence of unspecified functionality, whether accidental or intentional.

In particular, this document expands and unifies the testing requirements that are stated in the *Life Cycle Management Plan* [1], the *Configuration Management Plan* [2], and the *Software Development Standards* [3].

The kinds of tests that must be performed during and after a product has been built are specified in the *Life Cycle Management Plan*, and there is an assumption that the reader is somewhat familiar with the contents of that document. This Quality Assurance (QA) Plan emphasizes requirements, restrictions, standards, responsibilities, etc., for these required tests. Specifically excluded from this plan, however, are the formal and semi-formal work, code correspondence, and covert channel analysis. In addition, there will need to be independent re-testing and penetration testing performed. It is also recognized that quality means more than just source code testing (such as conformance to documentation standards, correct spelling, etc.), but those issues are currently covered in other documents.

Figure 1 shows the high-level flow of development and testing up to and including a quality assurance audit on the engineering tests.

This QA Plan has been written under the assumption that a small team of engineers is developing a product. Larger organizations and larger projects will likely require modifications to this plan to fit their needs.

Rationale: The flow shown in Figure 1 has all the product testing performed by the engineering staff prior to acceptance into the CM Repository, instead of performing all the testing on objects that are generated from the CM Repository. Unlike a low assurance development environment where the official code repository is the same as the development code repository, the CM Repository in this designed environment is physically separated from the developers, and changes are controlled by the CCB. In this environment, if the product tests were performed only after the CM Staff had generated objects from a CCB-approved source tree, then bugs found that late in the testing process would require the overhead of another submission to the CCB to fix the bug(s). Another alternative approach that was not adopted because of inefficiencies was to have the engineers perform the product tests before CCB submission, and then have the CM Staff repeat those tests after the product had been regenerated from CCB-approved items. The adopted approach has the engineering staff perform all the tests prior to CCB submission, and then have the CM Staff regenerate the objects and then compare those regenerated objects to the submitted objects to verify they are identical.

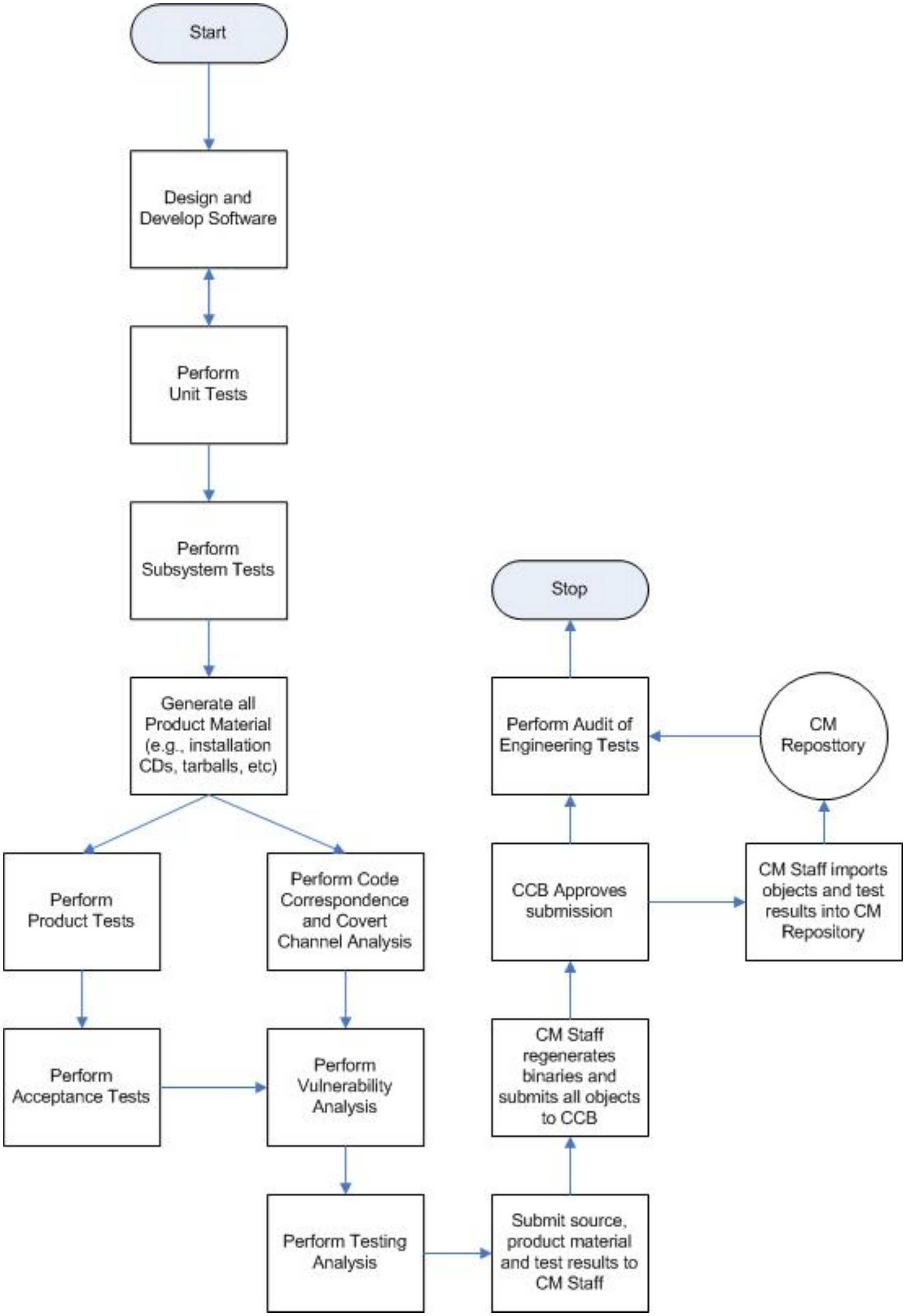
Start
Design and Develop Software
Perform Unit Tests
Perform Subsystem Tests
Generate all Product Material (e.g., installation CDs, tarballs, etc)
Perform Product Tests
Perform Code Correspondence and Covert Channel Analysis
Perform Acceptance Tests
Perform Vulnerability Analysis
Perform Testing Analysis
Submit source, product material and test results to CM Staff
CM Staff regenerates binaries and submits all objects to CCB
CCB Approves submission
CM Staff imports objects and test results into CM Repository
CM Reposttory
Perform Audit of Engineering Tests
Stop

Figure 1 Development and Testing Flow

2 General Testing Requirements

All design documents shall be written in a format that allows for easy reference by their associated test plan. For example, a high-level design document shall uniquely identify design criteria such that the low-level design can cross-reference its design back to the high-level design upon which it depends.

All test plans (e.g., Product Test Plan) shall be written in a format that allows for easy reference to all test objectives. For example, a Subsystem Test Plan shall state its testing objectives in a format that shall allow the writer of the Subsystem Tests documents to reference the individual testing objectives stated in the plan, such as numbering all test objectives from T-1 to T-n.

Test plans shall describe the purpose or goal of each test. If there is a required order for the tests, e.g., to establish a desired state, then the rationale for the ordering shall be documented in the test plan for future reference. Security-relevant tests shall be identified in the test documentation.

When test documents are written, each individual test shall be identifiable in some fashion, for example by using a numbering scheme. A mapping shall then be given that shows the correspondence of each individual test to a test objective. This mapping shall show complete coverage of all test objectives.

Test documents shall be written with enough detail for independent parties to re-establish the same test environment and conditions, and allow such a tester to conclude that the test input was appropriate, and that the output produced a success or failure.

Test documents shall minimally be comprised of a procedure. This procedure shall describe the steps to be performed to execute a given test, and the expected outcome of the test. "The results of all test cases shall be documented." [3] Space shall be provided in the testing procedures to allow the tester to describe the observed outcome, and other comments as necessary.

Testers shall be required to sign and date the completed procedure. If more than one person performs the tests, then those portions performed by each individual shall be noted. The completed, signed and dated procedures shall be submitted as evidence when the tested CI is submitted to the Configuration Control Board (CCB) for Configuration Management (CM), and shall be maintained as evidence by CM personnel.

"Testing strategies and test cases shall cover the following:

- Positive behavior

  Testing needs to show that all required functionality works as specified.

- Negative behavior

> Testing needs to show that obvious undesired behavior is not present. For example, it is not enough to test whether an authorized subject can access an object; the testing shall also show that an unauthorized subject cannot access an object.
>
> Where possible, all error conditions shall be tested to ensure that the condition is detected, and that the specified reaction is seen (e.g., the proper error code is returned)." [3]

When a test requires source code to be written, the mapping from the testing source code to a specific test or tests shall be documented in the source code. Because the size of the development group is assumed to be small, "it shall ... be acceptable for the higher-level tests to be written and administered by someone who wrote some of the modules comprising the subsystem and product. In such a case, a peer review of the higher-level test code shall judge whether the tests are complete." [3]

The tests themselves do not need to be formally tested, but it is up to the CI Leader to determine the level of effort to show that the tests function properly. In addition, tests shall be reviewed as specified in Section 3, and shall comply with the coding standards. [3]

The CI leader shall avoid the appearance of a conflict of interest during the testing phases, e.g., having the developer who wrote the code do all the testing for the code, unless it is expressly permitted in the *Life Cycle Management Plan*.

The CI Leader has leeway to manage a CI, including how the CI source code and test code are handled before, during and after testing. However, the CI Leader shall demonstrate with a high degree of confidence that the CI source code being submitted to the CCB produced the object files that are being submitted with the source code, and that the test code being submitted was the code used to perform the testing. The test plan and test procedures shall provide controls and evidence to support that assurance.

If source code is changed before it is submitted to CM but after it has been tested (e.g., due to a bug fix, or a problem found during peer review), then it shall be retested. If source code is changed after a peer review has been performed, then the peer review shall be performed again, minimally for the portion that changed.

3 Requirements for Test Plan and Test Development

This following sub-sections describe the requirements for the various test plans and tests that must be developed.

3.1 Acceptance Plan and Acceptance Tests

"The purpose of the Acceptance Plan is to provide the strategy for testing a product before it is considered ready for delivery [to Integration]. The Acceptance Plan uses the Requirements Definition as its input." [1]

“The Acceptance Tests are the tests, procedures, check-lists, etc., that implement the Acceptance Plan.” [1] The Acceptance Tests shall be reviewed by a non-author of the tests with at least the same technical ability. The review shall minimally verify that all the test objectives in the Acceptance Plan have all been covered, and that all required tests have been implemented.

The plan shall stipulate that the official Acceptance Tests (i.e., those that shall be kept as evidence) shall be executed against objects that have not been modified since their respective Subsystem Tests have been completed.

The Acceptance Plan and the Acceptance Tests shall be maintained in the same Configuration Item (CI) that contains the Requirements Definition. Evidence of the required reviews shall be included when the CI is submitted to the CCB for acceptance into CM.

Requirements for the actual execution of the Acceptance Tests are provided in Section 4.

3.2 Product Test Plan and Product Tests

“The Product Test Plan is the strategy for testing the completed product for compliance with the designed external product interfaces, as documented in the Functional Specification.” [1] The Product Test Plan shall be written in a manner that shows how all the external interfaces in the Functional Specification are covered by the tests.

“The Product Tests are the tests, procedures, check-lists, etc., that implement the Product Test Plan.” [1] All tests shall be reviewed by a non-author of the tests with at least the same technical ability. The review shall minimally verify that all the test objectives in the Product Test Plan have been covered, and that all required tests have been implemented.

The plan shall stipulate that the official Product Tests (i.e., those that shall be kept as evidence) shall be executed against objects that have not been modified since their respective Subsystem Tests have been completed.

The Product Test Plan and Product Tests shall be maintained in the same CI that contains the Functional Specification. Evidence of the required reviews shall be included when the CI is submitted to the CCB for acceptance into CM.

Requirements for the actual execution of the Product Tests are provided in Section 4.

3.3 Subsystem Test Plan and Subsystem Tests

The Subsystem Test Plan “is the strategy for testing the external interfaces of the completed subsystem, as documented in the High-Level Design.” [1] The Subsystem Test Plan shall be written in a manner that shows how all the external interfaces in the High-Level Design are covered by the tests. The plan shall require a full regeneration of subsystem files prior to testing.

“The Subsystem tests are the tests, procedures, check-lists, etc., that implement the Subsystem Test Plan.” [1] All tests shall be reviewed by a non-author of the tests with at least the same technical ability. The review shall minimally verify that all the tests specified in the Subsystem Test Plan have been covered, and that all required tests have been implemented.

The Subsystem Test Plan shall describe how the objects being tested will be protected from modification during and after the execution of the Subsystem Tests, up to and including their submission to the CCB. The plan shall provide a process for verifying the integrity of the objects at any time prior to their submission to the CCB.

The Subsystem Test Plan and Subsystem Tests shall be maintained in the same CI that contains the associated High-Level Design. Evidence of the required reviews shall be included when the CI is submitted to the CCB for acceptance into CM.

Requirements for the actual execution of the Subsystem Tests are provided in Section 4.

3.4 Unit Test Plans and Unit Tests

“The Unit Test Plan is the strategy for testing each module.” [1] The Unit Test Plan shall be written in a manner that shows how all the module interfaces in the Low-Level Design are covered by the tests. The plan shall provide a strategy for showing adherence to the Low-Level Design as well as development of any additional tests that are deemed necessary after a peer review of the source code has been performed.

“The Unit Tests implement the Unit Test Plan.” [1] The Units Tests shall be reviewed by a non-author of the tests with at least the same technical ability. The review shall minimally verify that all the tests specified in the Unit Test Plan have been covered, and that all required tests have been implemented.

The Unit Test Plan and Unit Tests shall be maintained in the same CI that contains the modules they are designed to test. Evidence of the required reviews shall be included when the CI is submitted to the CCB for acceptance into CM.

Requirements for the actual execution of the Unit Tests are provided in Section 4.

4 Composition and Test Execution Requirements

This section describes the requirements for performing the various tests.

4.1 Unit Tests

“Modules are implemented from ‘the bottom up’, meaning that the independent modules are implemented first. After each module is implemented, it must undergo unit testing. These bottom-layer modules then form the foundation for implementing modules in the next layer up. These modules then undergo unit testing before continuing to the next layer. And so it continues until all the modules have been implemented and unit tested.” [1]

“It is acceptable for the author of a source code representation of a module to write and administer the unit tests. This allows the module to be tested before other dependent modules are written.” [3]

Because perfection may not be possible, some problems may be identified during the unit tests that cannot be easily resolved, or are postponed for later action by the CI Leader. A flaw report for such problems shall be submitted in a timely fashion.

4.2 Subsystem Tests and CCB Submission

“After all the modules of a subsystem have been completely implemented and unit tested, the subsystem must be tested according to the Subsystem Test Plan.” [1] “It is possible for a subsystem to be completely implemented before the subsystems it depends on are implemented. In this situation the subsystem can be tested, as long as the dependent subsystems are emulated in some kind of 'test harness' with sufficient expected behavior of the unfinished subsystems. It is then possible to have a subsystem implemented, tested and baselined before these dependent subsystems are implemented. However, there must be a balance struck between the time and effort to implement such a test harness, and the time it will take to wait for the actual subsystems to be completed.” [1]

All flaws that are discovered during subsystem testing shall be promptly reported.

A subsystem can be submitted to the CCB for CM after it has “passed all its Subsystem tests” and “undergone appropriate reviews”. [1] However, for this QA Plan to work efficiently, all CIs that are expected to change for a given release shall be submitted simultaneously, as coordinated by the Project Manager. (See the sidebar *Submitting a Subsystem to the CCB* for the rationale for this requirement.) The submission shall include both source and generated files.

A subsystem may, in fact, be submitted to the CCB with known problems. The Project Manager, in consultation with the CI Leaders, determines whether the subsystem is “good enough” for submission. All known problems shall be identified by their unique flaw identifier in the submission paperwork, along with a justification for postponing action to a later release of the product.

When an identified subsystem CI is received by CM personnel for submission to the CCB, CM personnel shall perform a recompilation on a system separate from the CM Server. To verify that the generated files used for subsystem testing correspond to the submitted source files, the generated files shall be compared to the equivalent files on the submitted media. If they do not match, the CCB submission is returned to the CI Leader.

Comparing Object Files

Depending on the header format, an object file header may contain information, such as a time stamp, that is different for each re-compilation, even if nothing has changed in the source files. In such situations there shall be a tool or manual procedures that “blocks out” such portions during the binary comparison of object files.

After the CI has been accepted by the CCB, the source files are checked into the CM repository. Another recompilation is performed within the CM system. The prospective official generated files are once again compared with the submitted object files to ensure that the baselined objects were the tested and approved objects.

Submitting a Subsystem to the CCB
There are two general approaches for submitting a subsystem to the CCB, each with its own advantages and disadvantages. These approaches are described below.

- Submitting subsystems as they are finished.

  After a subsystem has completed its tests and reviews there is nothing to prevent it from being submitted to the CCB. One risk to this approach is that flaws may be discovered when higher-level subsystems exercise lower-level subsystems during their testing. That may happen anyway whether the subsystem is submitted to the CCB or not, but an additional submission would be required following any corrections, adding administrative overhead.

  Another problem with this approach is the necessity of submitting subsystems in a lower-level order, such that a higher-level subsystem cannot be submitted prior to any subsystems it depends on. If this was not done, then the recompilation or subsequent object file comparisons may fail because dependent files (e.g., header files) that have changed have not been submitted yet.

- Submitting all subsystems at once.

  One approach is to wait until all subsystems have been tested and reviewed before they are submitted to the CCB, and to submit them simultaneously to the CCB for approval. This would eliminate the problems described above. This is the approach used in this QA Plan.

4.3 Product Composition and Product Tests

“After all the subsystems have passed their tests, they are composed into a working version of the product. The Product Tests are applied against this composition.” [1]

Note that there shall exist a CI that includes the responsibility of doing product-wide “makes”, composing various subsystem object files into composed executable files, if necessary. These composed files are submitted as part of the CI CCB submission and verified in the same fashion as subsystem object files, namely, a build is performed by CM to verify that the CM-generated files are equivalent to those that exist on the submitted media. This product-wide build shall be performed from scratch (i.e., a “make clean; make”).

Prior to the execution of the Product Tests it shall be shown that the objects being tested have not been modified since their Subsystem Tests have been completed. Evidence of this integrity shall be maintained with the Product Tests and submitted to the CCB. In a larger organization this testing would be done by a separate group of engineers dedicated to testing, after the software engineers were "finished".

Flaws found during the Product Tests shall be reported immediately.

4.4 Acceptance Tests

"The last testing step is to validate that the finished product meets the requirements specified in the Product Definition by performing the Acceptance Tests." [1] The Acceptance Tests shall be performed using the same objects that were used for the Product Tests. Prior to the execution of the Acceptance Tests it shall be shown that the objects being tested have not been modified since their Subsystem Tests have been completed. Evidence of this integrity shall be maintained with the Acceptance Tests and submitted to the CCB.

Flaws found during the Acceptance Tests shall be reported immediately.

4.5 Vulnerability Analysis and Testing Analysis

"Another activity that must be performed on the product is a vulnerability analysis. This analysis takes the flaws found during testing (and other means), and ensures that the flaws cannot be used to violate the enforced security policies in some way. The outcome of this activity is recorded in a document known as the Vulnerability Analysis." [1]

"In conclusion, after all tests have been performed, an analysis must be made to show that the testing included sufficient depth and breadth. The outcome of this activity is recorded in a document known as the Testing Analysis." [1] This report shall also show that the ordering of tests did not conceal potential flaws.

4.6 Quality Assurance Audit

After a product submission has been approved by the CCB a quality assurance audit is performed. The audit minimally includes re-testing a sub-set of the Product Tests. The amount of re-testing is determined by the Project Manager.

References

[1] P. C. Clark, C. E. Irvine, T. Levin, and T. D. Nguyen, "Trusted Computing Exemplar: Life cycle management plan," Naval Postgraduate School, Monterey, CA, Tech. Rep. NPS-CAG-14-002, Dec. 2014.

[2] P. C. Clark, C. E. Irvine, T. Levin, T. D. Nguyen, and D. Warren, "Trusted Computing Exemplar: Configuration management plan," Naval Postgraduate School, Monterey, CA, Tech. Rep. NPS-CAG-14-003, Dec. 2014.

[3] P. C. Clark, C. E. Irvine, T. Levin, T. D. Nguyen, and D. Shifflett, “Trusted Computing Exemplar: Software development standards,” Naval Postgraduate School, Monterey, CA, Tech. Rep. NPS-CAG-14-007, Dec. 2014.

[THIS PAGE IS INTENTIONALLY BLANK]

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Ft. Belvoir, Virginia

2. Dudley Knox Library, Code 013 2
   Naval Postgraduate School
   Monterey, California 93943

3. Research Sponsored Programs Office, Code 41 1
   Naval Postgraduate School
   Monterey, California 93943

4. Paul C. Clark 1
   Naval Postgraduate School
   Monterey, California 93943

5. Dr. Cynthia E. Irvine 1
   Naval Postgraduate School
   Monterey, California 93943

6. Thuy D. Nguyen 1
   Naval Postgraduate School
   Monterey, California 93943